Binary Search Trees

7/16/2009

Important Dates

• Project 2 due
  – DESIGN, DESIGN, DESIGN!!!
  – You may have 0 or 1 partner.
  – NO EXCEPTIONS!
  – Friday 7/24/2009 – 10pm
• Project 2 – time to work in lab
  – Monday 7/20/2009
• Midterm Review
  – Saturday 7/26/2009 – 1-4pm in 306 Soda
• Midterm 2
  – Tuesday 7/28/2009 – 5-6pm in 10 Evans

Topics

• Maximally balanced trees
• Big-Oh of tree operations
• Binary Search Tree remove
• Stacks
• Map

Maximally Balanced?

3

5

10

18

15

7

5

10

15

13

18

2
Another way to implement trees!

Maximally Balanced Trees – WHY?

A Better Tree Implementation
(don’t use the 0 index)

How many elements are in a balanced tree? Assume $h =$ height

$\begin{align*}
n &= 2^{h+1} - 1 \\
&= \frac{1}{2} \log_2 \left( \frac{n+1}{2} \right) \\
&= h - 1 \\
&= \frac{\log_2 n}{2}
\end{align*}$

What is the run-time for find in this tree?

$\begin{align*}
\log_2 2^h &= h \\
\log_2 n &= \frac{h}{2}
\end{align*}$
What is the run-time for find in this tree?

Running Times

- In a perfectly (full) balanced binary tree with height/depth \( h \), the number of nodes \( n = 2^{h+1} - 1 \).
- Therefore, no node has depth greater than \( \log n \).
- The running times of \( \text{find()} \), \( \text{insert()} \), and \( \text{remove()} \) are all proportional to the depth of the last node encountered, so they all run in \( O(\log n) \) worst-case time on a perfectly balanced tree.

Running Times

- What’s the running time for this binary tree?
- The running times of \( \text{find()} \), \( \text{insert()} \), and \( \text{remove()} \) are all proportional to the depth of the last node encountered, but \( d = n - 1 \), so they all run in \( O(n) \) worst-case time.

Running Times

- Binary search trees offer \( O(\log n) \) performance on insertions of randomly chosen or randomly ordered keys (with high probability).
- Technically, all operations on binary search trees have \( \Theta(n) \) worst-case running time.
- Algorithms exists for keeping search trees balanced. e.g., 2-3-4 trees.

Running Times

- The Middle ground: reasonably well-balanced binary trees
  - Search tree operations will run in \( O(\log n) \) time.
- You may need to resort to experiment to determine whether any particular application will use binary search trees in a way that tends to generate balanced trees or not.

Binary Tree Remove
http://www.cs.jhu.edu/~goodrich/dsa/trees/btree.html

Delete a node given a key, if a node exists.

1. Find a node with key \( k \) using the same algorithm as \( \text{find()} \).
2. Return \( \text{null} \) if \( k \) is not in the tree;
3. Otherwise, let \( n \) be the first node with key \( k \). If \( n \) has no children, detach it from its parent and throw it away.
**Binary Tree Remove**

4. If \( n \) has one child, move \( n \)'s child up to take \( n \)'s place. 
   \( n \)'s parent becomes the parent of \( n \)'s child, and \( n \)'s child becomes the child of \( n \)'s parent. Dispose of \( n \).

**Deletion**

5. If \( n \) has two children:
   - Let \( x \) be the node in \( n \)'s right subtree with the smallest key. (the inorder successor) Remove \( x \); since \( x \) has the minimum key in the subtree, \( x \) has no left child and is easily removed.
   - Replace \( n \)'s entry with \( x \)'s entry. \( x \) has the closest key to \( k \) that isn't smaller than \( k \), so the binary search tree invariant still holds.

**Stack**

- **push**: put something on top of the stack
- **pop**: take something from the top of the stack
- **peak**: look at the thing on the top of the stack

**Stack Based Pre-Order Traversal**

**Map is an Interface**

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jonathan</td>
<td>Hot pink</td>
</tr>
<tr>
<td>Kaushik</td>
<td>Green</td>
</tr>
<tr>
<td>David</td>
<td>Blue</td>
</tr>
<tr>
<td>George</td>
<td>Green</td>
</tr>
<tr>
<td>Colleen</td>
<td>Yellow</td>
</tr>
</tbody>
</table>

**Map Interface**

- **get**: returns the value for that key
- **put**: takes a key and a value and adds them to the map.
- **containsKey**: returns a boolean